LOCALPKI: A User-Centric Formally Proven Alternative to PKIX

Jean-Guillaume Dumas, Pascal Lafourcade, Francis Melemedjian,
Jean-Baptiste Orfila, Pascal Thoniel

LocalPKI: A User-Centric Formally Proven Alternative to *PKIX*

Jean-Guillaume Dumas*[1], Pascal Lafourcade†[2], Francis Melemedjian‡[3], Jean-Baptiste Orfila§[1] and Pascal Thoniel¶[3]

[1]Université Grenoble Alpes, CNRS, LJK, 700 av. centrale, IMAG - CS 40700, 38058 Grenoble cedex 9, France
[2]Université Clermont Auvergne, LIMOS, Campus Universitaire des Cézeaux, BP 86, 63172 Aubière Cedex, France
[3] NTX Research SA, 111 Avenue Victor Hugo, 75116 Paris, France

Abstract

A public-key infrastructure (PKI) binds public keys to identities of entities. Usually, this binding is established through a process of registration and issuance of certificates by a certificate authority (CA) where the validation of the registration is performed by a local registration authority. In this paper, we propose an alternative scheme, called LocalPKI, where the binding is performed by the local authority and the issuance is left to the end user or to the local authority. The role of our third entity is then to register this binding and to provide up-to-date status information on this registration. The idea is that many more local actors could then take the role of a local authority, thus allowing for an easier spread of public-key certificates in the population. We also prove our scheme's security with Tamarin an automatic verification tool of cryptographic protocols.

1 Introduction

Public Key Infrastructure (abbreviated *PKI*) allows to bind the user identity with his public key. On internet, the *PKIX* standard is widely used. Almost all major internet players authentify their servers with certificates via the *TLS* (Transport Layer Security) protocol. In this paradigm, end entities rely on *Certificate Authorities* (*CA*) which deliver *X*.509 certificates containing, at least, the user's identity and public key and a validity period. These certificates are then signed by the *CA*. Then, to use the public key of an entity, a user verifies the associated certificate: checking validity period, correctness of signature etc. To use long enough validity periods, a revocation mechanism has then to be set up. The most deployed solutions for this are *Online Certificate Status Protocol* (*OCSP*) [18] and *Certificate Revocation List* (CRL) [6]. In the first case, the user

*Jean-Guillaume.Dumas@univ-grenoble-alpes.fr
†Pascal.Lafourcade@udamail.fr
‡Francis.Melemedjian@ntx-research.com
§Jean-Baptiste.Orfila@univ-grenoble-alpes.fr
¶Pascal.Thoniel@ntx-research.com

sends a validity query of the owner's certificate to an *OCSP* responder (either maintained by the certificate's issuer, or well known to the user). In the second case, lists of revoked certificates must be regularly updated and published into *CRL Distribution Points* (*CDP*). Users access these *CDP* and check that the owner's certificate is not present inside the list. In the end, depending on the obtained certificate status, the owner is authenticated or not by the user.

The current state of the art is quite substantial, because of various security and efficiency requirements, mostly depending on the environment where the PKI must be set-up (internet, industrial architecture, power limited devices...). In terms of efficiency, solutions to reduce communication and computation costs of the revocation checks arose, such as *H-OCSP* [14] or *Delta-CRL* [6]. Moreover, the needed trustfulness in the *CA* has been intensively studied in order to reduce the impact of malicious behavior. Indeed, in *PKIX*, a compromised *CA* is able to ruin the entire authentication mechanism by delivering illegitimate certificates. Then, solutions based on public logs of *CA*'s action emerged, with for instance *Certificate Transparency* [12]. In this paradigm, certifications are included in append-only log structures: the log maintainer is able to prove that a specific certificate is present into the structure, and that each new added certificate has only extended the previous structure. The *Accountable Key Infrastructure* [10] exploits public-logs for the certificate management and distributes the trust between several entities. In [17], the authors provide a solution combining public revocation and a more largely distributed trust (using users' browsers) with the *Certificate Issuance and Revocation Transparency*. More recently, the *Attack Resilient Public-Key Infrastructure* (ARPKI) [2] proposes a PKI where clients choose several authorities (*CAs* and log maintainers) involved into the global process. There, each of these authorities supervises the others' behavior: as a consequence, a single non compromised entity is sufficient to prevent attacks. The solution described in [21] with the *Distributed Transparent Key Infrastructure* (DTKI), provides security even in the case where all these entities are corrupted. Notably, the security of ARPKI and DTKI has been formally proven using the automatic cryptographic protocols verification tool Tamarin [13, 19]. Most of these advanced solutions provide enhanced security to the cost of some complexity in the procedures. In a practical set up these solutions involve several authorities, which must be involved in the management of technical services (this is particularly true for the management of worldwide append-only logs). Furthermore, these schemes rely on a *CA*'s signature of certificates in order to guarantee the binding between the public key and the identity.

Another issue of *PKIX* is the certificates cost that still remains a problem. In order to reduce this cost and to simplify the implementation complexity, the project *Let's Encrypt* emerged to democratize the implementation of server certificates (`https://letsencrypt.org/`). Differently, the deployment of people certificates has never really been carried out on a large scale and still few end-users have certificates.

Contributions. In this paper, we formally describe and prove the security of a PKI called LocalPKI, based on the *PKI* 2.0 paradigm [4]. The *PKI* 2.0 project and its instances such as LocalPKI are seeking to democratize the attribution of people certificates, like what *Let's Encrypt* is doing for server certificates. They are essential for secured transactions: authentication, digital signature, confidentiality. LocalPKI removes three known bottlenecks of *PKIX*: remote delivery, cost of certificates and their complexity of use. The idea is to replace *CA* signed certificates by user self-signed certificates. However, contrary to *PGP* [22], trust is not given by users but by an authority. This authority, a combination of notaries and local actors in our setting, guaranties the binding. Indeed, after registration of a certificate owner by a notary, other users will be able to verify the authenticity of this certificate via a request. For a private verification, a possibility is to send a request to the notary, similar to an *OCSP* procedure: the notary's response depends on the looking up of uniquely recorded legitimate users in his database. For a public verification, a possibility is to provide hashed and signed subsets of the database, similar to NSEC3 [11] (or NSEC5 [20]) records within DNSSEC [3]. Also, as in *PKIX*, registration and authentication do not need be performed by the same entities. In *PKIX* the former step can be performed by a *Registration Authority*. In LocalPKI, this

entity is known by the notary and must be close to the user. It can be a technical service, just like a classical registration authority, but it can also be a local actor or a business service, closer to users. We have in mind banks, postal offices, mobile network operators, delivery points, university offices, for example, and more generally any actor used to check identities. For the user, he can get a free certificate (with possibly paid options available) near his home or work place with the security of face-to-face enrollment (identity verification). For local registration desks, it would often be in its own interest to deliver people certificates to its members or clients. These member/client certificates will enable them to authenticate and perform online operations, for instance securely signing contracts. Moreover, the involved entities, except for the notaries, do not really need any technical knowledge. Overall, our proposition, LOCALPKI, is a first instance of the PKI 2.0 paradigm offering an alternative to *PKIX*. LOCALPKI is able to provide the same services as *PKIX*, but using a user-centric approach and without the need of a signature by an authority for each certificate. Further, in this paper, we also provide a security analysis of our protocols, using Tamarin [13, 19] and show that the implementation of the LOCALPKI in a practical environment can be realized using existing *PKI* tools.

Organization of the paper. In section 2, we start by defining entities involved in LOCALPKI, and we make a high-level comparison with *PKIX*. Then in Section 3, we formally describe all the protocols involved, i.e. the registration, authentication and revocation mechanisms. Section 4 is devoted to the deployment of the architecture, using existing tools and solutions. Finally, in Section 5, we define the required security properties and an associated formal model for LOCALPKI. Those enable us to formally prove the security of LOCALPKI, using Tamarin.

2 General Description

LOCALPKI is a set of protocols which fulfills all the requirements of a public-key infrastructure (*PKI*): registration of a new user, authentication of registered users, revocation of certificates, renewals, etc. We start by introducing each entities' role.

Entities. There are three different entities in LOCALPKI: *Electronic Notaries* (*EN*), a *Local Registration Authorities* (*LRA*) and *users* (or *End Entities*). The *Electronic Notary* might be seen as the root *CA* in a classical PKI architecture. He manages the databases containing registered users. The *Local Registration Authority* represents the intermediate entity between the user and the notary. It is somewhat like a *Registration Authority* in a classical PKI, but in LOCALPKI it is closer to users than to the *CA*. In practice, the *LRA* could be an agency close to the user, such as the user's insurance company, his bank, the postal office, etc. Those agencies usually already have the abilities to check identities. The *LRA* is registered by some *EN*, and the identity checks are performed during the recording of a new user. Finally, users represent the entities who want to authenticate or be authenticated by others.

Comparison with *PKIX*. The main differences between LOCALPKI and *PKIX* are:

- In LOCALPKI registration authorities do not need to be security experts. Therefore they can be closer to users and allow more widespread deployment of the use of certificates in every day life (see § 3.2).
- Certificate creation is done by the *CA* in *PKIX*; in LOCALPKI it is done by the *LRA* while the notaries store and make this decision available (see § 3.2).
- The default authentication mode in *PKIX* is buffering via *CRL*; while default authentication mode in LOCALPKI is interactive, somewhat like *OCSP Online-Certificate Status Protocol* (see § 3.3.1).
- The alternative authentication mode in *PKIX* is *OCSP*, while LOCALPKI can also propose an alternative buffering mechanism called *Certificate Verification Lists* (*CVL*), described in Section 3.3.2.

From a closer look at both protocols execution, the major difference is in the signature of the user's

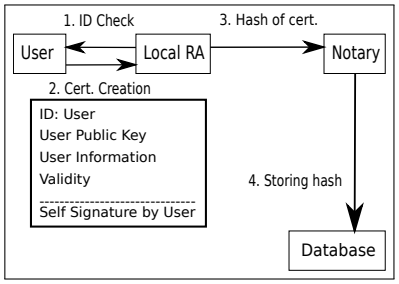

Figure 1: LOCALPKI registration.

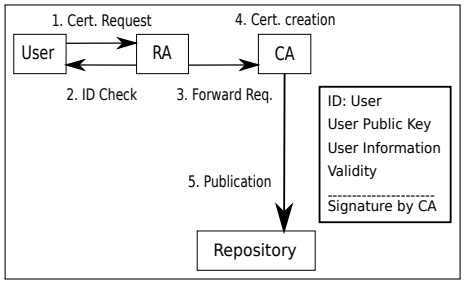

Figure 2: *PKIX* registration.

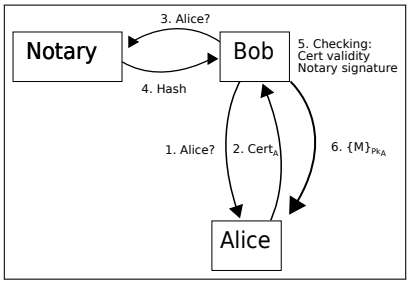

Figure 3: LOCALPKI end entity interactive authentication.

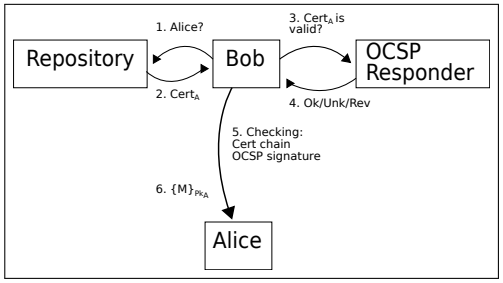

Figure 4: *PKIX* end entity interactive authentication.

certificate: in *PKIX*, the *CA*'s signature is present whereas in LOCALPKI only the self signature made by the user is required. As shown in Figure 1, certificate creation is realized in the 2^{nd} step by the owner instead of the 4^{th} one by the *CA* for *PKIX*, as shown in Figure 2. Furthermore, the registration authority forwards a certificate request to the *CA*, while the *LRA* only sends the certificate's hash to the notary. Finally, in LOCALPKI certificates are not directly published since the *EN*'s database only contains hashes.

Hence, interactive authentication is also different as shown in Figures 3 and 4: with LOCALPKI, owners have to provide their certificates to users beforehand, and then the users interact with notaries in order to be convinced of the certificate's validity; with *PKIX* users can recover certificates in a local repository and then interacts with OCSP responders to be convinced of the certificate's validity. The full protocol, in particular the buffering mechanism (the *CVL*, also called public mode within LOCALPKI), is detailed next.

3 Protocol Description

We now give some notations and then formally define the LOCALPKI protocols.

3.1 Notations

We denote by Pk_A (resp. Sk_A) the public key (resp. private key) of a user A. We write $\{m\}_{Pk_A}$ (resp. $\{m\}_{Sk_A}$) the action of ciphering (resp. signing) a message m with the public key Pk_A (resp. the private key Sk_A). Hashing a message m is written $H(m)$, with H the hash function. The concatenation of two messages m_1 and m_2 is denoted $m_1||m_2$. Generally, the notation O_A is used to express the belonging of the object O (e.g.,

a certificate) to the user A. We denote by $X_{509}()$ a function which takes user's information and returns them into the X_{509} certificate format without any signature.

3.2 Registration of a New User

In order to be known within the infrastructure, and to allow him to be authenticated by the others, a user first needs to be enrolled into the system: this step is called *registration*.

The first phase is the new key pair generation by the user. Then, he interacts with a *LRA* to start the registration process. In LOCALPKI the *LRA* should be physically close to the user, so that they can meet in person. The user begins by providing ID proofs according to the established security policy (e.g., a visual check of the ID card). Once the identity check succeeds, the user gives his public key to the *LRA*. Next, the authority generates a field equivalent to the *ToBeSigned* (TBSCert) in the $X.509$ certificates, containing at least: the user's ID, his public key, a *Serial Number* (*SN*), a validity period and the URL of the notary associated with the *LRA*. The *SN* has been obtained by the *LRA* from previous exchanges with the *EN*. The latter is in charge of the *SN* generation, and communicates to each supervised *LRA* a specific range of *SN*. For the next step, the user hashes the previously generated $TBSCert$ and signs the digest: the result is called *SI* (*Signature Id*). Afterwards, by using the previously provided public key, the *LRA* is able to recompute the digest and then to check the correctness of the signature. At this step, the user has proven his knowledge about the associated private key to the *LRA*. The final registration phase is performed by the *EN* who registers the unique couple (SN, SI) (this is a simplified version of what is called a "public key ownership certificate" in [4]). For this, the *LRA* ciphers the couple using the *EN*'s public key, and sends the result along with its signature of this message (i.e., $\{H(SN,SI)\}_{Sk_{LRA}}$). In the end, the registered user owns a certificate *Cert* containing the $TBSCert$, and in particular the *SN* and the *SI*, while the *EN* has added the associated couple (SN, SI) to his database. The complete process is detailed in Algorithm 1 and schemed in Figure 5.

Algorithm 1 Registration of Alice

Require: The *LRA* owns *a priori* serial numbers, provided by a trusted electronic notary *EN*.
Ensure: Identity check (by the *LRA*) and registration of Alice into the *EN* database.

1: Alice generates his public key Pk_A.
2: Alice $\rightarrow$ *LRA*: Pk_A
3: *LRA* checks information and identity of Alice.
4: *LRA* $\rightarrow$ Alice: Serial number (SN_A), notary URL (URL_{EN}), validity.
5: Alice generates a $X.509$ certificate $TBSCert_A$ (completed with URL_{EN} and SN_A), and computes $SI_A = \{H(TBSCert_A)\}_{Sk_A}$
6: Alice $\rightarrow$ *LRA*: $TBSCert_A || SI_A$
7: *LRA* checks SI_A (PoK of the Alice private key).
8: **if** Verification OK **then**
9: *LRA* $\rightarrow$ *EN*: $\{SN_A || SI_A\}_{Pk_{EN}} || \{H(SN_A || SI_A)\}_{Sk_{LRA}}$
10: **end if**
11: *EN* decides to add Alice to his database.

3.3 Authentication

Once the owner of the certificate has been correctly registered, other users could authenticate him i.e., they are ensured that the used public key is indeed the owner's one. The authentication process in LOCALPKI

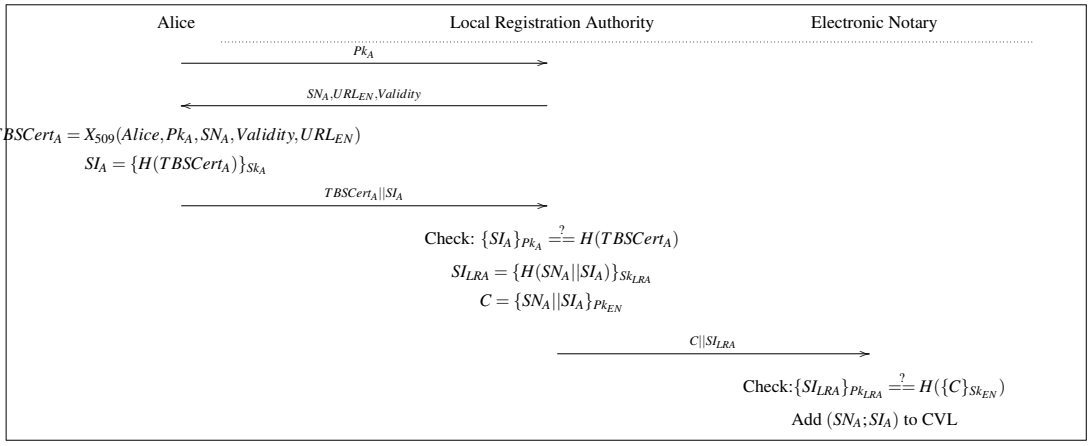

Figure 5: Registration of Alice.

can be realized in two different ways. First by using a private mode, where only the *EN* knows the full database containing registered users. In this case, the user requests the *EN* about the validity of the owner's certificate, and no more information is revealed. The second possibility is to apply the public mode (also called buffering mode), where the *EN* shares parts of his database with the user, who makes the validity verification by himself. In both cases, the user must be able to first authenticate the *EN* using the URL provided in the certificate. As in *PKIX*, the mechanism used in LocalPKI is the *trust anchors* [16]. A trust anchor contains certificates of the trusted notaries. Then, users are able to retrieve information on notaries, in particular their associated public key. In the following, we detail both the private and public modes.

3.3.1 Private Mode.

The specificity of the private mode resides in the database of registered users, which is only known by the *EN*. Then the validity check of the owner's certificate by another user (the *verifier*) is made by interacting with the *EN* at verification time. First the verifier gets the certificate from the owner. Then, the verifier has two possibilities: he can check the owner's self signature by himself (Algorithm 2) or delegate also this task to the notary (Algorithm 3).

In the first case, the *Authentication Request* (*AR*) only contains the couple (SN, SI) and a nonce R. In the second case, the AR is made of the complete certificate and a nonce. In any case, the entity verifying the signature has to extract the public key, the *SI* and the *TBSCert* from the certificate. Then, he hashes the *TBSCert* and applies the cipher on the *SI* using the public key. The verifier also checks that the *EN* associated to the URL contained in the certificate belongs to his trust anchor, otherwise authentication cannot be realized. Then, he sends the correct AR (depending on the choice of the signature verification) to the indicated *EN*. During the next step, the *EN* looks for the given IDs in his database. If they are found, he responds positively, otherwise he gives a negative answer. The complete message consists of the previous answer, the nonce R, the couple (SN, SI) and the signature by the notary of all the previous contents. Finally, the verifier checks the *EN*'s signature of the answer using the public key extracted from the *EN*'s certificate (stored in the user trust anchor). An example of authentication can be found in Figure 6.

Algorithm 2 Certificate check in private mode (self signature verification)

Require: Alice gets the certificate from Bob. She wants to check the validity of the certificate.
Ensure: Authentication of Bob to Alice if the certificate is correct, failure otherwise.

if $\{SI_A\}_{Pk_B} \stackrel{?}{=} H(TBSCert_B)$ **then**
  Alice: $R_A \xleftarrow{\$}$
  Alice $\rightarrow URL_{EN}$: AR=$SN_{Bob}||SI_{Bob}||R_A$
  if $(SN_{Bob};SI_{Bob}) \in$ Database **then**
    $Rep =$ "OK"$||AR$
  else
    $Rep =$ "$Unknown$"$||AR$
  end if
  $URL_{EN} \rightarrow$ Alice: $Rep||\{H(Rep)\}_{Sk_{EN}}$
  Alice checks the response signature, and authenticate (or not) Bob.
end if

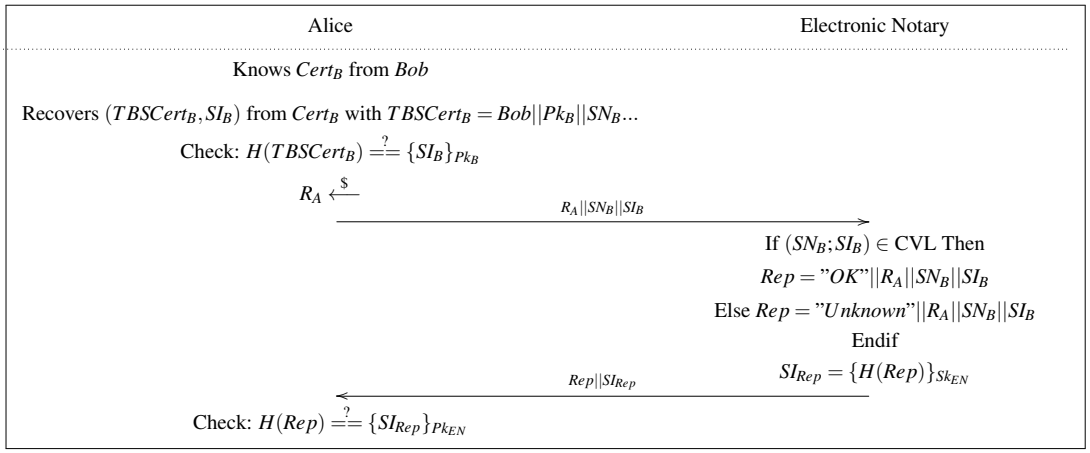

Figure 6: Authentication of Bob by Alice (in private mode).

3.3.2 Public Mode: Certificate Verification List.

In the public mode, checking the owner's certificate validity is realized by the verifier. He first obtains the owner's certificate. Then he must check that the certificate's signature (i.e., the *SI*) is consistent with the information contained in the *TBSCert* (i.e., the public key and the *SN*). After having checked that the *EN* belongs to his trust anchor, the verifier sends a request to the *EN* designated by its URL in the certificate. This request asks for the *Certificate Verification List* (*CVL*) i.e., the content of the database storing the couple (SN, SI) of previously registered users. To exchange the *CVL* with the user, the *EN* sends a signature of the *CVL* in addition to the list itself. This also ensures its integrity. Finally, the verifier checks the signature of the *EN*, and then verifies that (SN, SI) belongs to the *CVL*. Of course, just like a *CRL*, a *CVL* can be locally stored (buffered) and reused in a certain time interval without any refreshment. Just like for a *CRL*, a typical refreshment rate for a *CVL* could be in days. An example of public authentication is given in Figure 7.

Even if the public mode reduces the number of operation realized by the *EN*, it implies a non-negligible communication cost. In order to reduce it, the idea is to divide the database into subdomains. Indeed, the

Algorithm 3 Certificate check in private mode (delegate signature verification)

Require: Alice gets the certificate from Bob. She wants to check the validity of the certificate.
Ensure: Authentication of Bob to Alice if the certificate is correct, failure otherwise.

1: Alice: $R_A \xleftarrow{\$}$
2: Alice $\rightarrow URL_{EN}$: AR=$Cert_{Bob}||R_A$
3: **if** $\{SI_A\}_{Pk_B} \stackrel{?}{=} H(TBSCert_B)$ **then**
4: **if** $(SN_{Bob}; SI_{Bob}) \in$ Database **then**
5: $Rep =$ "OK"$||AR$
6: **else**
7: $Rep =$ "$Unknown$"$||AR$
8: **end if**
9: **else**
10: $Rep =$ "$Wrong\ Signature$"$||AR$
11: **end if**
12: $URL_{EN} \rightarrow$ Alice: $Rep||\{H(Rep)\}_{Sk_{EN}}$
13: Alice checks the response signature, and authenticate (or not) Bob.

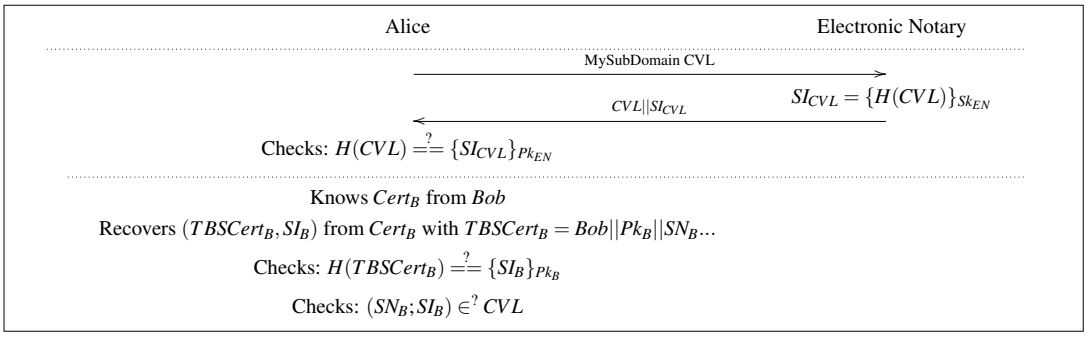

Figure 7: Authentication of Bob by Alice (in public mode).

EN is in charge of the generation of the *SN* given to the *LRA*. Hence, the database is intrinsically divided into *LRA* subdomains. Then, the verifier could buffer only the *CVL* associated to a subdomain, and hence the communication cost between the client and the *EN* is reduced to this subdomain size.

3.3.3 Comparison between Public, Private Modes and *PKIX* Mechanisms.

LOCALPKI is designed to be used by default in the private mode, offering a lower communication cost and an always up-to-date database. Nevertheless, the *CVL* are interesting in the case where an online interaction is not always guaranteed. In comparison with mechanisms used in *PKIX*, the private mode can be assimilated with *OCSP* and *CVL* with *CRL*. Initially, *OCSP* was not designed to resist against replay attacks. In a later version, a counter-measure has been added to the `responseExtensions` field (see [18, § 4.4.1]). However, as discussed in [18, § 5], this solution is not required by the norm. In the private mode of LOCALPKI, replay attacks are countered by default.

In *PKIX*, the complete list of revoked certificates must be shared. Thus, the communication cost is large. A way to reduce these large broadcasts, on solution can be to use δ-*CRL*'s. Similarly we proposed to allow

subdomain CVL's. In the case of the *CVL*, only the subdomain part containing the certificates need to be exchanged to have a correct authentication. But, unlike δ-*CRL*, a subdomain *CVL* is fail-safe. Indeed a user looking only in a partial *CRL* can miss that a certificate has been revoked with another reason, and still use it. On the contrary, if a valid certificate is not present in a subdomain *CVL*, it just cannot be used.

Furthermore, *CRL* based solutions expose users to false positive authentications. Since a *CRL* is not continuously updated, recently revoked certificates could still be considered valid. By using a white-list strategy like *CVL*, verifiers may not succeed in authenticating newly registered users. However, once again, obtaining a false negative is usually a safer fail than a false positive. Besides that, only valid certificates are stored. Then, in case of authentication failure, the verifier cannot know the reason (revocation, unregistered user).

3.4 Revocation

A public key infrastructure should provide a solution to revoke certificates before the end of their validity period. This allows for instance to manage lost or compromised key pairs. In LOCALPKI, a certificate revocation can easily be done by the certificate's owner or by the *LRA*, via a request to the *EN*. In both cases, this request is signed and contains the owner's certificate along with a `Revoke` message. In case of a request from the owner's certificate, the signature acts as a proof of knowledge of the private key. In case of a request by the associated *LRA*, the notary has anyway to accept registrations from this *LRA* (for instance within a range of allowed serial number values). This same range can be used to guaranty that the *LRA* is allowed to revoke this serial number. Hence, in both cases, the *EN* verifies the signature and if the couple (SN, SI) is present in his database, he simply removes the entry. Thus, an authentication attempt using this revoked couple fails, since the entry has been removed from the *EN* database. Then the renewal procedure is simple: after the revocation, the user has to enter a new registration process with his *LRA*.

Algorithm 4 Certificate revocation

Require: Alice correctly registered in the *EN* database. $X \in \{LRA, User\}$
Ensure: The certificate revocation of User
1: $X \rightarrow EN$: $SI_{Rev} = Cert_X || \{H(\text{"}Revoke\text{"} || (SN_{User}; SI_{User}))\}_{Sk_X}$
2: *EN* checks signature
3: **if** Verification is OK and $(SN_X; SI_X) \in$ DataBase **then**
4: *EN* removes $(SN_X; SI_X)$ from the database.
5: **end if**

4 Deployment

LOCALPKI has the advantage to be easily deployed from an existing *PKIX*. Actually, each of these requirements can be satisfied by adapting the current standards of *PKIX*. In the following, we present how to realize it.

- First of all, certificates employed in LOCALPKI can be based on *X.509v3* certificates [6]. Indeed, both PKI, LOCALPKI and *PKIX*, share the same kind of identification data (TBSCert). Then, the *CA*'s signature is replaced by the user's signature and the *SN* can be stored in the `serialNumber` field.
- The communications with the notaries during authentication in the private mode, Figure 6, can be set up using the *OCSP* norm [18]. Within the *OCSP* request, the *SN* replaces the current `serialNumber` in

the `CertID` sequence, and the *SI* is stored in the `signature` field of the optional `Signature` sequence. Note that other fields in `CertID`, such that the `issuerNameHash` and `issuerKeyHash`, could be left empty since this information is either irrelevant or redundant with the *SI*. The nonce R_A (see Figure 6 and Algorithms 2 and 3) can be stored into the *OCSP* `requestExtensions`.

- Similarly, the notary's answer can also follow the *OCSP* response format, where all the latter fields are also present.
- In the public mode, Certificate Verification Lists (*CVL*) can be managed just like Certificate Revocation Lists (*CRL*). For example, a *CVL* could be published on the *EN* websites, and can be stored in local repositories. Moreover, if an organization with subdomains is required, e.g., each range of *SN* represents a subdomain, the *Delta-CRL* indicator could used [6], as well as the `BaseCRLNumber` field which could represent for us an equivalent Base CVL number field.
- The revocation requests within LocalPKI and *PKIX* are almost identical. Thus, LocalPKI may use the *Certificate Management Protocol* [15] directly for revocations.
- Finally, *PKIX* requires trust anchors [16] to be deployed, e.g., within the store of the users' browsers or OS's. Communications between *LRA*s and notaries, also require an anchor mechanism and that of *PKIX* can also be used directly.

Therefore, existing tools like *OpenSSL* allow for all entities to generate keys, certificates, authentication and revocation requests or responses. The technical setup of LocalPKI is mainly restricted to the management of the databases. This is delegated to the *EN*, who are thus also in charge of maintaining the *PKI* availability. *LRA*s are in charge of communications with the notaries, that is mainly exchanging serial numbers, and of the face to face identity verifications.

Differently, users have several possibilities, depending on their expertise.

- Expert users, first generate themselves their own key pair. Then they request a serial number, *SN*, through the *LRA*, in order to create and sign their certificate. Finally they give the associated *SI* to the *LRA* who will forward it to the notary.
- An intermediate possibility, is for the user to only generate a key pair. The *LRA* will then take charge of the certificate creation and provide means for the user to sign it with his private key (for instance a usb port and a keyboard so as to type the password deciphering the private key stored in a usb device).
- The *LRA* can also create fresh key pairs on the fly and provide everything to the user.

Part or all of the latter two possibilities could even be realized through a dedicated web site. Therefore, the only technical requirement for the *LRA* is the use of tools like *OpenSSL*, in order to help the users registration and the creation of a user-friendly associated API.

5 Security Analysis

The main goal of LocalPKI is to make authentication of users possible when using asymmetric cryptography. However, the protocol must provide other security guarantees. In the following, we define the desired security properties. Then, we show how we model the protocol in the automatic cryptographic protocols verification tool called *Tamarin Prover*, in order to prove these security properties.

5.1 Security Properties

The first property that is required is that the protocol must be *correct*, i.e., if a person has been correctly registered into the database and her certificate is still valid, then he must be correctly authenticated. The underlying property correspond to classical authentication property and is thus about correct identity checks.

Vice versa, the *soundness property* can be defined as follows: an adversary who has not been registered cannot be authenticated. This property implies that an adversary cannot forge a certificate considered as valid and cannot impersonate a valid one. A reformulation of the soundness property is that authentication at time i_2 implies a registration at time i_1 with $i_1 < i_2$.

In [2], the authors defined the *Connection Integrity* as follows: if a user establishes a connection with another one, then the user communicates with the legitimate owner of the private key. In others words, in the case where registration has been correctly done (i.e., by honest participants), no adversary can possibly know the private key of the honest owner.

Moreover, the protocol must ensure some secrecy properties. Indeed, a protocol execution must not reveal any sensitive information: once authenticated, the adversary cannot know the message.

To summarize, by assuming that *LRA* and the *EN* are trusted, LOCALPKI verifies the following security properties:

- *Correctness*: if a user has been correctly registered and is not revoked then he must be correctly authenticated ;
- *Soundness*: if a user has been authenticated, then he must have been registered before and he has not been revoked before;
- *Connection Integrity*: if a user is correctly registered, then the adversary does not know his private key.
- *Secrecy*: once a user is authenticated, the messages sent to him cannot be learnt by the adversary.

5.2 Tamarin Prover Modeling

In order to prove these security properties, we use *Tamarin Prover* an automatic security protocol verifier [13]. This tool implements a protocol verification technique for unbounded number of sessions. Adversaries are defined according to the Dolev-Yao model [7]. This means that adversaries can extract information from every exchanged messages. The usual perfect encryption hypothesis is also assumed in this tool (i.e., the adversary does not learn any information from a encrypted message if he does not own the associated private key). Moreover, we use classic equational theories provided by default by Tamarin for the crypto-operations like ciphering or signing messages. In Tamarin, the protocols are modeled using multiset rewriting rules. The rules are composed of facts, which are used to model the local knowledge of a participant, such as the reception of a message, or the generation of a fresh number or the emission of a message. Then, each player action is implemented as a `rule`. A rule rewrites a fact into another one, and is eventually labeled in order to trace the realized actions. For example, a rule rewriting a fact composed of a message and key into another state containing the cipher of the message with the previous key could be labeled as *Cipher*. Facts could be persistent (denoted with a `!` before its name), which means that it can be reused arbitrarily often by rules. Otherwise facts are said to be linear and can be used only once. Once all the protocol rules are defined, we model the desired properties as `lemmas`. A lemma is a first-order logic sequence applied on the label of previously defined rules. They contain quantifiers (∀: `All`, ∃: `Ex`) and logical connectives (`&`, `|`, `not`, `==>`), along with timepoints (declared with `#`, and employed with `@`). For example `Ex sna sia #i. Bob_Auth_Alice(<sna,sia>) @i` means that the event *Bob Authenticates Alice* using variables `<sna,sia>` occurs at time `i`.

Now, we detail our model and the properties proved. We also show that trust hypothesis on entities are mandatory, just like, e.g., trust hypothesis are required on CAs in *PKIX*, by describing attacks found by the tool if any of these hypothesis is removed.

All the Tamarin source files can be found at: `http://localpki.forge.imag.fr/`. See the associated *Makefile* for details on how to generate proofs and attacks.

5.3 LOCALPKI Tamarin Model

We consider a protocol execution where only one notary, one *LRA*, one registered user (Alice) and one verifier (Bob) are represented, which is enough according to a well-known theoretical result [5] that proves that for secrecy and authentication properties only one intruder is enough and at most one honest participant per role. Our model of this execution comprises the registration of Alice, her authentication by Bob followed by an exchange of a message, and the revocation of her certificate.

In the classic model of *PKIX*, key pair generations are done by generic persistent facts, instantiated with different terms. These facts bind the identity of the entity with its public key to represent the trust anchors, i.e., other entities have the correct association between the public key and the identity. Thus, this method is usable for modeling the trust anchors of the *LRA* and the *EN*. In the case of Alice, we need to explicitly define the self-signed certificate described in the Algorithm 1. Moreover, we assume that the Alice's certificate is public.

We employ two types of communication channels, depending on the situation: a real-life meeting and insecure channels. Since the registration must be realized during a real-life meeting, we assume that it is not subject to any intruder attack. Therefore, our model expresses the information exchanged between the *LRA* and Alice in a totally private manner (i.e., the adversary is not able to learn or modify any information for this exchange). All others communications are exposed to eventual wiretapping.

The registration of the couple (SN_A, SI_A) in the database is represented as a state which associate the knowledge of (SN_A, SI_A) to the identity of the *EN*.

Finally, for revocation, since Tamarin is working on execution traces, we must represent the removal of the certificate from the database. For this we have employed a tag into a persistent fact to express that the revocation is definitive.

5.4 Security Properties

First, we explicit security properties i.e., lemmas, in the case where all entities are supposed honest but in presence of an intruder. As defined in the Section 5.1, we have implemented lemmas about correctness, soundness and secrecy of the protocol.

Soundness. The first lemma (`soundness`) proves the soundness of the LOCALPKI. The couple `<sna,sia>` represents *SN* the *Serial Number* and *SI* the *Signature Id* of Alice. Each label in the lemma represents the moment where the actual event occurs. The idea is to prove that for all possible couples (sna, sia) related to the Alice's private key `ltkA`, if Bob has successfully authenticated Alice at time `i` (`Bob_Auth_Alice()`), then it means that Alice has been previously registered (at time `j`, by the notary, `EN_Reg_Alice()`), and that if the certificate has been revoked (`Cert_Is_Revoked()`), then it was at an earlier time `k`:

```
lemma soundness:
  all-traces
 "All EN B sna sia #i.
  Bob_Auth_Alice(B, <sna,sia>, ltkA) @i
  ==>
 (Ex #j. EN_Reg_Alice(EN, <sna,sia>, ltkA)@j
  & (j<i)) &
 (All #k.
  Cert_Is_Revoked(EN, <sna,sia>, ltkA)@k ==> i<k)"
```

Correctness. The second lemma (`correctness`) ensures that our model is correct, i.e., there exists an execution where Alice is registered, Bob authenticates Alice and the certificate is not revoked. Thanks to

this lemma, we check that our model realizes the protocol steps in a correct order. Other lemmas in our modeling also provide sanity checks, for example to show that there exists a trace where the certificate has been correctly revoked.

```
lemma correctness:
  exists-trace
 "(Ex EN B sna sia ltkA #i #j.
  EN_Reg_Alice_(EN, <sna,sia>, ltkA) @i
  & Bob_Auth_Alice(B,<sna,sia>, ltkA) @j
  & not(Ex EN #l.
  Cert_Is_Revoked(EN, <sna,sia>, ltkA) @l & i<l))"
```

Secrecy. In the secrecy lemma (`secrecy`), we ensure the secrecy of exchanged messages using a LOCALPKI based authentication. Then, we prove in Tamarin that the message (denoted `x`) should not be in the adversary knowledge, written as `K()` at any time.

```
lemma secrecy:
  all-traces
  "All x #i. Secret(x) @i ==> not(Ex #j. K(x)@j)"
```

Connection Integrity. Similarly, in the connection integrity lemma (`connection`), we ensure that the secret key of Alice, `ltkA`, should not be in the adversary knowledge neither in the case where Bob authenticates Alice from her certificate nor in the case where the certificate has been revoked.

```
lemma connection:
  all-traces
  "(All EN sna sia ltkA #i.
    Bob_Auth_Alice(B, <sna,sia>, ltkA)@i
    ==> not(Ex #l. K(ltkA)  @l))
  & (All B sna sia ltkA #i.
    Cert_Is_Revoked(EN, <sna,sia>, ltkA) @i
    ==> not(Ex #l. K(ltkA)  @l))"
```

In other words, whatever subsequent messages sent by any user, Alice's private key remains private.

5.5 Trust assumptions

Next, we show that our trust assumptions in both the *LRA* and the *EN* are mandatory in order to preserve the security of the protocol (as is the case in *PKIX*). To show this we present attacks on the protocol where one of the entity is malicious i.e. its private key is given to the adversary.

Trust in the *EN*. In the case where the notary is corrupted, the secrecy is not verified because he is able to add a false key pair to his database. Then after the authentication of Alice by Bob with these keys, a wiretap between communications allows him to retrieve secret exchanged information. Soundness is also falsified: since the adversary has access to the *EN*'s private key, he acts as Alice has been previously registered into the *EN* database whereas she is not. In practice, this attack represents the possibility to the notary of giving to the person of his choice the trust to be authenticated. Concerning the connection integrity property, Tamarin finds an attack as soon as the *EN*'s key is leaked to the adversary. The attack is the following: after its initialization, the *EN* registers himself as Alice. This means that the adversary forges a certificate using Alice's identity, and uses its own private key to sign the certificate. At the next step, Bob authenticates Alice using this certificate, and the adversary knows the associated private key: therefore the connection integrity is broken. First, this attacks highlights the obvious forgery ability of a malicious notary. Second, it shows that the connection integrity property is well-defined. Indeed, such a forgery ruins the integrity, even if the

adversary has no knowledge of the Alice's actual private key. This proves that the *EN* should be a trusted entity.

Trust in the *LRA*. When the *LRA*'s private key is leaked, the tool does not find any attack on the soundness. From a practical point of view, this result makes sense: the *LRA* is only involved into the registration process. Then, even if he provides false information, Alice should be registered before an authentication process. However, by removing the restriction of a unique registration per couple (SN_A, SI_A), the adversary sends a revoked couple to the *EN*, in order to pass over the revocation. In practice, serials numbers are provided by the *EN* so that this attack is easily preventable, by marking the used ones. Secrecy is falsified because the adversary impersonates the *LRA* during the last step of registration, by signing false information (i.e., false key pair). Then, since Bob does not exchange messages with the correct Alice's public key, and a simple wiretap allows the adversary to retrieve secret messages. When the *LRA*'s private key is given to the adversary, the connection integrity is also broken. During this attack, Alice registers herself to the *LRA*. Then the *LRA* modifies Alice's certificate by including its own key pairs: the public one into the certificate, and the private one is used to sign the forged certificate. Then, the fake certificate is sent to the *EN*. Afterwards, suppose that the *EN*, for any reason, decides to revoke this certificate. At this point in time, the *EN* has succeeded in revoking the certificate while the adversary knows the associated private key. Hence, the connection integrity is broken. This attack shows the *LRA* ability to forge certificates. In conclusion, this proves that *LRA* should be a trusted entity.

Trust in Alice. When Alice is corrupted, repercussions on the protocol security are limited. It does not influence soundness property. This result is coherent: even with Alice's private key is given to the adversary, he cannot be authenticated without a previous registering. In practice, a malicious behavior from Alice could be an identity fraud during the registering if the *LRA* has been misled. However, this kind of attacks are out of the Tamarin scope, since ID checking cannot actually be modeled. On secrecy aspects, obviously, if the private key of Alice is leaked, the adversary is able to read the secret messages sent to Alice. This attack could correspond to the Alice's private key theft, where both Alice and the attacker are able to retrieve messages ciphered with the associate public key. Finally, if the private key of Alice is leaked, connection integrity is obviously broken, since the adversary can then use her key. Overall, we conclude that Alice does not need to be a trusted participant.

5.6 Security of the LocalPKI

From our security analysis, using the Tamarin prover, we have proven the following theorem:

Theorem 1. *If the notary and the local registration authority are not corrupted, then the* LocalPKI *security architecture is correct, sound, and preserves confidentiality and integrity.*

Our implementation consists of about 350 source lines of code to describe the model consisting of 20 rules and 4 lemmas. In order to reduce proof timings we have associated each entity to a defined `Role`. Moreover, we have enforced some actions to be unique (denoted as `Only_One`). These constraints are called `axioms` into the code. They avoid generic rewriting of rules either by associating a fixed string to an identity variable or either by implying timepoints equality in case of multiple uses of the same rules. Therefore, using a single core of an *i5 4590@3.50Ghz* with *8GB RAM*, we obtain very good performance for the security verification, as shown in Table 1.

Table 1: Timings of Tamarin's proofs of lemmas.

Lemma	CPU Time
Correctness:	4.7 s
Soundness:	4.4 s
Connection integrity:	10.9 s
Secrecy:	6.6 s

6 Conclusion

In this paper, we have proposed an alternative public-key infrastructure model, LOCALPKI. This model has been analyzed and its security formally proven using Tamarin. The main feature is that, unlike the *PKIX* standard, here user certificates are self-signed and only the binding of a certificate with an identity is signed by a third-party, a notary in LOCALPKI. Therefore, it is easier for users to use certificates within their local environment. Furthermore, the registration process can be transferred to the notaries by local businesses, which only have to handle identity verifications. We think that this could foster a wider spread of certificates among everyday end users. For this, notaries just have to maintain an accessible database of fingerprints. Hence, LOCALPKI is an alternative to the *PKIX* solution, providing similar security properties.

However, in some cases, the key management of LOCALPKI is better than *PKIX*. For example, bank customers have the possibility to look at their account from a website, sometimes using a certificate-based authentication. Managing a classical *PKIX* either requires a large internal department or a full delegation of trust to a *PKI* actor. By deploying the LOCALPKI solution, the bank still preserves a local registration, and thus trust and responsibility of its customers, and the technical part of ensuring their authentication is deported to notaries. Overall, the cost and the technical requirements for the company are reduced. Moreover, LOCALPKI also offers to small agencies the possibility to help users for their registration. The economics incentive would be to offer this service only at a moderate cost, where deploying a full *PKI* in every agency should be too expensive.

Also, as seen in the Section 4, the overall deployment of LOCALPKI can be made using only existing tools, which facilitates the process.

As further work, legal aspect on the shared responsibility of the *LRA* and the *EN* in LOCALPKI should be studied, since both entities play an important role into the registration and authentication processes. Our first idea is that the *EN* could register the identity of the *LRA* sending information about new users and store this information also within the database. Thus, if this user is recognized as malicious, the *EN* could blame the *LRA*.

Further work also includes a study about adapting the secure trust computations described in [8]. They compute a global trust by using the trust evaluation of each certification authorities towards the others. This allows to obtain a better hint about the trust given in *CAs*, instead of being based only on the trustfulness given by trust anchors. This method could be relevant to the LOCALPKI architecture, since it is also using the trust anchor mechanism.

We also plan to realize a study of whether this model could also simplify identity-based approaches like certificateless PKI [9, 1].

Finally, just like *Let's Encrypt* offers certificates enabling HTTPS (via SSL/TLS) for websites, we plan to set up a web site offering the possibility to create self-signed LOCALPKI certificates for any user.

References

[1] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless public key encryption without pairing. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2005.

[2] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack resilient public-key infrastructure. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 382–393, November 2014.

[3] Jason Bau and John C. Mitchell. A security evaluation of DNSSEC with NSEC3. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.

[4] S. Bouzefrane, K. Garri, and P. Thoniel. A user-centric PKI based-protocol to manage FC2 digital identities. *IJCSI International Journal of Computer Science Issues*, 8(1):74–80, January 2011.

[5] Hubert Comon-Lundh and Véronique Cortier. Security properties: two agents are sufficient. *Sci. Comput. Program.*, 50(1-3):51–71, 2004.

[6] Dave Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[7] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.

[8] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Private multi-party matrix multiplication and trust computations. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016)*, pages 61–72, 2016.

[9] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, pages 272–293, Berlin, Heidelberg, 2003. Springer-Verlag.

[10] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. Accountable key infrastructure (aki): A proposal for a public-key validation infrastructure. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 679–690, New York, NY, USA, 2013. ACM.

[11] Olaf M. Kolkman, Matthijs Mekking, and R. (Miek) Gieben. DNSSEC Operational Practices, Version 2. RFC 6781, December 2012.

[12] Ben Laurie, Adam Langley, and E Kasper. Certificate authority transparency and auditability. *white paper*, 22, 2011.

[13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[14] Jose L. Muñoz, Oscar Esparza, Jordi Forné, and Esteve Pallares. H-ocsp: A protocol to reduce the processing burden in online certificate status validation. *Electronic Commerce Research*, 8(4):255, 2008.

[15] Martin Peylo and Tomi Kause. Internet X.509 Public Key Infrastructure – HTTP Transfer for the Certificate Management Protocol (CMP). RFC 6712, September 2012.

[16] R. Reddy and C. Wallace. Trust anchor management requirements. RFC 6024, RFC Editor, October 2010.

[17] Mark Dermot Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[18] Stefan Santesson, Rich Ankney, Michael Myers, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.

[19] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE Computer Society, 2012.

[20] Jan Vcelak, Sharon Goldberg, and Dimitrios Papadopoulos. NSEC5, DNSSEC Authenticated Denial of Existence. Internet-Draft draft-vcelak-nsec5-03, Internet Engineering Task Force, September 2016. Work in Progress.

[21] Jiangshan Yu, Vincent Cheval, and Mark Ryan. DTKI: A new formalized PKI with verifiable trusted parties. *Comput. J.*, 59(11):1695–1713, 2016.

[22] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.